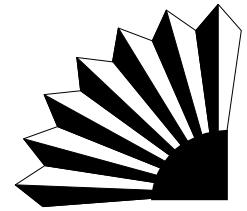


# the Technical Broadcast



Published by the Department of Information Services

Fall/Winter 1996

## Year 2000 Project Methodology

 by SIG Sub-Committee Methodology

### RENOVATION -- APPLICATION CODE CHANGE

**T**his document represents the collective thinking of the State of Washington Year 2000 Special Interest Group (SIG) sub-committee working on a common high level approach to making state computer applications capable of processing in the new millennium. The Year 2000 SIG is composed of Information Technology Professionals designated as Year 2000 project coordinators by their individual state agencies. The high level conversion approach described below assumes that a detail inventory and analysis of an agency's computer applications has been done and that these applications have been grouped into one or more "application sets" that are ready for renovation. The conversion approach below (or methodology) covers code conversion, data conversion, and the related testing associated with these conversion changes.

Prior to using this year 2000 renovation methodology, the following steps should have been performed:

- An agency specific definition of the meaning of "year 2000 compliant" is documented.
- All programs, copybooks, JCL, PROCs, databases, and files have been inventoried.

The application inventory has been divided into groups which will be converted, tested, and placed back into production as a unit ("application set").

*(Continued on page 2)*

<http://www.wa.gov/dis/CSD/tbfirst.htm#Broadcasts>

#### Inside...

<a href="#">COBOL/370 and LE/370, Article Six</a>	<a href="#">12-25</a>	<a href="#">How to Find Unused GDG Base Catalog Entries Using MASTERCAT</a>	<a href="#">30-34</a>
<a href="#">Natural Getmain Threads in CICS</a>	<a href="#">26</a>	<a href="#">Billing of Tape and Disk Data Sets in the S/390 MVS Environment</a>	<a href="#">35-37</a>
<a href="#">DFSORT Year 2000 Enhancements</a>	<a href="#">27-29</a>		

---

# Year 2000 Project Methodology

*(Continued from page 1)*

- All application program source code matches the current production object code and has been placed in maintenance status.
- A detail conversion plan with schedule is complete for the “application set” about to be converted.
- An overall cost estimate of conversion is complete.
- The business risk assessment is known.
- A detail cross reference of jobs, files/databases, copybooks, and programs is known.
- All internal and external data bridging requirements with other interfacing applications have been identified.
- Application code change (i.e. expansion of year date fields from 2 to 4 digits) is the remedy of choice.

The choice to change the application code should be made after carefully examining the alternative approaches available to allow the application to function properly into the next millennium. Alternatives reviewed should have included: code modification, re-engineering, replacement, and retirement.

Large scale code modification is best done using a “factory-like” approach. Assembly line processes and a high level of automation are used to expand all date fields within the code to a full 4-digit year. Additionally, standard date processing routines (known to be cross-century compliant) will replace existing routines. Temporary or permanent bridging and/or date windowing may be inserted to affect interfacing to existing file/database formats. The input transaction (terminals and batch) and output reporting (reports and terminal screens) date formats will not expand unless specifically needed to eliminate ambiguity. The greatest productivity is achieved using this methodology and limiting the number and complexity of code changes. Limited change activity reduces the coding and especially the testing time.

This code modification process takes advantage of the fact that the application code changes can be made and tested using test facilities provided under the current Washington State Department of Information Services (DIS) shared operating environments (V1RA, V2DL, or V4YM).

*(Continued on page 3)*

---

# Year 2000 Project Methodology

*(Continued from page 2)*

DIS is currently planning to provide base level cross-century operating system environment (MVS, security, database, on-line, utilities, etc.) support when vendors can provide cross-century compliant software levels. A minimum level of application re-testing should be planned to validate the operating system supports agency application systems properly after being IPL'd to certain "key" future century dates (i.e. 01/01/2000, 02/29/2000, etc.).

Several very large and/or complex applications exist in multiple state agencies which probably cannot be converted in a single "application set". The methodology was designed to accommodate these very large applications. The sub-committee's consensus was greatest flexibility and least interface complexity will, be achieved by separating the program code change process from the database/file date field expansion process. This separation will be accomplished by inserting temporary bridges into the programs to allow access to the non-compliance date data for some period of time until the actual databases and files have finally been converted. This allows independent schedules to be developed for the two different processes. Being able to schedule the database/file expansions separately from program modification greatly reduces the complexity associated with managing large numbers of application to application interfaces.

The following generic code modification plan should allow maximum productivity across the multiple affected state agencies while minimizing complexity. This methodology will be supported with tools and expertise provided by DIS and agency experiences to further enhance productivity as the process of code conversion proceeds over the next few years. Various alternative optional scenarios which explain how to use this methodology are documented following the discussion of the generic "worst case" scenario. The model plan is as follows:

## MODEL PLAN (BATCH)

### Phase I (Source Code Conversion)

**Objective:** The output of this phase of the plan is to make all necessary changes to the application code without the need to change related databases and files. File and database changes will occur during the next phase of this plan. Date fields within the program are expanded to include a full 4-digit year (i.e. CCYY instead of just YY) for all date handling processes. A standard set of century compliant date routines replace internal date processing routines. Windowing and/or bridging techniques accomplish the interface requirements with existing files and databases where needed.

*(Continued on page 4)*

---

# Year 2000 Project Methodology

*(Continued from page 3)*

## ASSUMPTIONS:

1. Expand all date fields containing 2-digit year to digits to include the century indicator (i.e. CCYY).
2. Replace all called date routines with year 2000 compliant standard date routines (i.e. Platinum Technologies).
3. Examine in-line user coded date routines and make decision to convert to standardized routines based on complexity and impact on testing.
4. Insert standard windowing routines as necessary for interfacing to databases and files (including input and output files).
5. Use bridge routines to bridge from/to databases/files.
6. Leave output reports/screens unchanged (2-digit year) except where required to remove ambiguity.
7. Leave input dates as 2-digit years except where required to remove ambiguity.



*(Continued on page 5)*

---

# Year 2000 Project Methodology

*(continued from page 4)*

## PROCEDURE:

1. Change Source Code (as per above assumptions)

Use automated tools wherever possible. No functional changes should be necessary or encouraged.

2. Unit Test (if necessary)

If a high degree of automation was used successfully to change the source, then unit testing may not be necessary. Move the code into a larger group for regression testing instead.

3. Regression Test (using current century date)

Use the regression test tool (developed during the Vanilla migration project<sup>1</sup> and updated for the Year 2000 project) to assist with this process. Save the data for later processing in the subsequent phases.

4. Move to Production

The applications should be cross-century compliant at this time, but not formally proven yet. The proof will occur in later phases.

5. Stop

## Phase II (Database/file Conversion)

**Objective:** The purpose of this phase is to expand the format and content of the date fields in databases and files to be cross-century compliant, and to verify that actual future date processing is successful. All application sets associated with these databases and files must have been modified according to Phase I (above) to make them cross-century compliant prior to date field expansion. In order to verify that the applications can correctly process using future year 2000 dates, additional testing is required. Date fields within the databases/files must be expanded and actually populated with “future dates” using an algorithm appropriate for the application, and the programs will be rerun using simulated “key” future dates (i.e. 12/31/1999, 01/01/2000, 02/29/2000, etc.).

*(Continued on page 6)*

---

# Year 2000 Project Methodology

*(Continued from page 5)*

## ASSUMPTIONS:

1. Related programs were made cross-century compliant in Phase I.
2. Archive/history files will not be converted. This implies that permanent bridging and/or windowing will be necessary to access this data from certain programs.
3. A limited number of “key” year 2000 dates have been selected which, if processed correctly during the system tests, will validate that the application systems are cross-century compliant.

## PROCEDURE:

1. Disable Bridges to Test Databases/Files

Turn off switches for internal program bridges and remove external step bridging in job streams. The physical code can be removed later after all other migration activities are completed during a clean-up phase.

2. Convert Test Databases and/or Files (for each test case)

Expand and reformat the various date fields such that 2-digit years are now 4-digit years. Use automated utilities where possible. Base test databases/files on the regression test files from Phase I. Use a combination of date field expansion and “time warping” (adding a fixed amount of time to every date field in each database/file, or some other appropriate algorithm) to build a test case for each “key” year 2000 date.

3. Set Up for Test Date Simulation (for each test case)

The date simulation tool will allow you to specify jobs/steps that will receive a specific date when they request the current system date. This information is provided separately by the tester before the job is run using function provided by the date simulation tool.

*(Continued on page 7)*

---

# Year 2000 Project Methodology

(Continued from page 6)

4. Perform System Tests (for each test case)

Capture the output from this run for comparison against a base case.

5. Compare Test Run Output Against Expected Output

If you are using the updated regression test tool (from the Vanilla migration project) you would compare the “P1” output after it has been converted to 4-digit year and has been time warped to match this test case against the test output just collected in step 4 above.

If outputs match, finish remaining test cases or go on to step 6. If outputs don’t match then fix program and re-test.

*--> Back to step 2 until finished with all cases*

6. Disable Bridges to Production Databases/Files

Leave the bridges necessary to access archive/history data.

7. Convert Production Databases and/or Files

Use the same procedures employed to convert the test databases/files. Just omit the “time warp” functions.

8. Move to Production

The application systems should now be verified year 2000 compliant.

9. Stop

(Continued on page 8)

---

# Year 2000 Project Methodology

*(Continued from page 7)*

## Phase III (Operating System Verification)

**Objective:** The purpose of this phase is to re-verify that critical client applications continue to process correctly on a cross-century compliant operating system platform which has been IPL'd to "key" future dates (i.e. 12/31/1999, 01/01/2000, 02/29/2000, etc.). (Note: DIS staff has begun and will continue to upgrade the system level software while agency application renovation activities proceed.) Agencies will perform re-verification testing following DIS' initial system compliance verification testing. All application sets and their associated databases and files must have been processed through Phase I and Phase II (above) to make them cross-century compliant. The compliant operating system environment will be IPL'd using "key" year 2000 dates identified by agencies and certain system tests will be performed by agencies on their mission critical application systems to re-verify that they continue to process correctly using these future dates.

### ASSUMPTIONS:

1. The most likely test environment is a non-shared TOTALLY independent hardware and software platform.
2. All libraries, databases, files, etc., must be off-loaded from the current environment and moved into the newly established platform test environment.
3. The timing of a platform test environment is dependent on the availability of cross-century compliant software from vendors.
4. DIS staff has primary responsibility to establish the platform test environment.
5. The DIS client has primary responsibility to transfer necessary application components to the platform test environment.
6. The SIG will act as an advisory group to work with DIS to determine schedule and special requirements for the platform test environment.

*(Continued on page 9)*



---

# Year 2000 Project Methodology

*(Continued from page 8)*

## PROCEDURE:

### 1. Build Operating System Software Compliant Environment

DIS will be working with the various vendors to build a cross century compliant operating system platform. Cross-century compliant vendor products will be placed into normal production LPARs through the course of normal maintenance over the next few years. When a high percentage of the vendor software becomes compliant a year 2000 compliant TEST environment will be established.

### 2. Establish Operating System TEST Platform (hardware and software)

At a point in time when a high percentage of the operating environment is cross-century compliant, DIS will establish a TEST system. It is envisioned that this environment must be independent and will not share ANY current DASD or TAPE.

### 3. DIS Operating System Environment Testing

DIS staff will run through test scenarios for all system components to verify correct system level processing using selected year 2000 dates.

### 4. Agency Operating System Environment Testing

Agencies will have access to the operating system test environment for re-verifying that sensitive application systems continue to process correctly using the new operating system components operating with year 2000 dates as they would if it were really in the future.

Note: Since there are no shared resources, client agencies should plan to offload, using tape, all necessary components such as program libraries, databases/files, JOB Libraries, CA-7, PROCs, etc.

### 5. Stop

*(Continued on page 10)*

---

# Year 2000 Project Methodology

*(Continued from page 9)*

The above model plan may be used in a number of different ways to accomplish different objectives. Phases I and II are where most of the agency activity is concentrated. Phase III testing is done toward the end of the project after all/most of the year 2000 conversion activity has been accomplished. In order to use the model plan most effectively, let's use some example scenarios to demonstrate its flexibility. In all scenarios listed below, you would put some selected applications through the Phase III activities to re-verify certain agency mission critical applications still operate properly in the new base system platform. The scenarios below refer to Phase I and II activities and assume you may run any applications you choose through Phase III.

## **SCENARIO 1 -- Compliance Certification**

Suppose you need to certify that a system you believe to be year 2000 compliant is truly ready. You either developed this application recently with year 2000 compliance in mind or you may have purchased a package claiming compliance. In any case, you do not need to change the code or the data, but just certify that it is ready. All you need to do is to proceed to Phase II, steps 2 through 5 to perform testing using the date simulation tool to help you simulate "key" year 2000 dates to validate that this system is year 2000 compliant.

## **SCENARIO 2 -- Smaller Application Sets (Preferred approach)**

Many of your systems may be small enough that you can change both the program code and the associated data at the same time. The determining factors are the amount of time necessary to keep the application set "frozen" for maintenance purposes, and the number of interfaces to other application sets. This is the preferred approach if you have the time. This will take the least amount of effort because you will not have to worry about extra temporary bridging and windowing necessary to access the non-compliance data. If you are able to do this, then you perform Phase I followed by Phase II activities except that you would not perform temporary bridge or window insertions while performing Phase I, step 1, and you would not perform Phase I, step 4 (Move to Production).

## **SCENARIO 3 -- Larger Application Sets**

Some of your application sets will be too large and/or complex to allow you to convert both program and data objects at the same time. This is the most complex case and requires using all three phases in sequence as outlined above. This approach should simplify the overall conversion while maintaining high efficiency.

*(Continued on page 11)*

---

# Year 2000 Project Methodology

*(Continued from page 10)*

## **SCENARIO 4 -- Windowing Only**

You may find that some of your applications do not require that the data be changed in the associated databases and/or files. For these applications you may wish to perform application code change only to allow proper handling of cross-century dates. This approach will employ the use of a windowing technique to properly process dates as they transition the century boundary. You should proceed through Phase I code changes and regression testing, and then through testing steps 3, 4, and 5 in Phase II for each “key” future date for this application.

For further information or questions, please contact Stan Davis at (360) 902-3151 or visit our year 2000 web site at: <http://www.wa.gov/dis/2000/y2000.htm>.



---

# COBOL/370 and LE/370, Article Six

 by Gary Duffield

**T**his is the sixth in a series of eight articles discussing features of the COBOL/370 programming language. This article discusses capabilities CONDITION/MESSAGE HANDLING, a feature of COBOL/370 running under LE/370.

Here's the idea: Instead of every program doing its own error response and message logic, why not establish a single process that spans across all the programs?

LE/370 supports this idea with Condition Tokens that are used to signal that a problem has been found. For example, in the previous article we explored one of the CALLABLE SERVICES that can return a Condition Token in the FEEDBACK-CODE (FC). Now, instead of having every program examine all possible bad FC values and decide what to do, why not just have them pass the token along to a CONDITION HANDLER? It can decide on the proper procedure, issue a message, and return control to the program, or abend, etc.

“Called” programs can signal conditions, but so can your main program. For example, your logic might test input fields for validity. Instead of dealing with invalid values itself, it could build a condition token and signal a condition. LE/370 itself can also signal conditions for events like system abends (S0C4, S0C7, etc).

You don't have to rely on the Condition Manager's default handling. You can write your own condition handler(s) that can be registered with the Condition Manager by your program. If the program subsequently signals a condition, the Condition Manager will give registered handlers first crack at the condition. The last one registered will get control first. It can handle the condition and direct the Condition Manager to return control to the program, it can choose not to handle it and percolate the condition token, or signal a new one. The handler previously registered would get the token next. So it goes until it percolates back up to the Condition Manager. If the process must terminate, it will first change the token to indicate “imminent termination” and pass it around again. This gives the handlers one last turn to issue a special message prior to shut down.

To put it simply: You can create a small collection of condition handlers for your shop. Then, instead of replicating error handling logic in every program, they can just register the appropriate handlers and signal conditions when appropriate.

*(Continued on page 13)*

---

# COBOL/370 and LE/370, Article Six

*(Continued from page 12)*

Working along with this is LE/370's concept of a central message library. Instead of every program's source containing the text of messages it might send, these are collected into a library. Then each program can ask LE/370 to send a certain message from the library to the message file (SYSOUT by default). These messages can be tailored to include what LE/370 calls "instance specific information" (text unique to that invocation). It just makes sense to have the message numbers derive from the condition tokens.

This article discusses the format of a condition token, writing and registering a condition handler, and signaling a condition.

## Format of a Condition Token

The token consists of 12 characters:

Positions	Contents	Description
1-2	Severity Identifier	0 = Informative 1 = Warning 2 = Error 3 = Severe Error 4 = Critical Error
3-4	Message Number	Can be combined with Facility ID to create a Message ID for message handling
5	Bits containing:	
	Case	Type of condition token
	Severity	Severity of condition
	Control	Origin of Facility ID
6-8	Facility ID	Identifies system signaling the condition CEE = LE/370 IGY = COBOL/370 You can create your own
9-12	Instance Specific Information	Unique information about this condition

*(Continued on page 14)*

---

# COBOL/370 and LE/370, Article Six

*(Continued from page 13)*

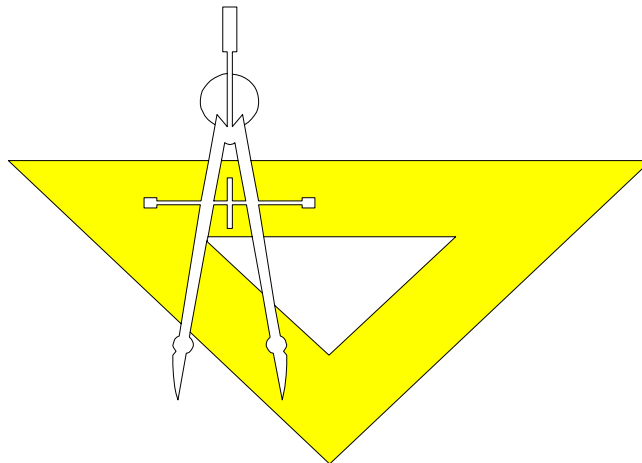
## Writing and Registering a Condition Handler

Who is going to invoke your condition handler? The LE/370 Condition Manager (as long as the handler has been registered with it so that it knows it exists -- more about that later). What information will the Condition Manager provide to your handler? The condition token, of course, and also four characters of data that were given it by the program that registered your handler. There are two things that the Condition Manager will expect back from your handler:

- (1) What action do you want it to take:
  - RESUME (return control to the program that signaled the condition)
  - PERCOLATE (give the token to the next handler in line)
  - PROMOTE (you want to change the token and give to the next handler)
- (2) The new condition token (if you choose promote)

Before we look at an example of the code for this, let's talk a little about those four characters passed to your handler by the Condition Manager. Four characters seem a pretty meager allotment. And the fact that its contents are established when the handler is registered and never change after that further seems to reduce its usefulness.

But remember in the previous article when we used a "pointer" to pass an address? Does that give you an idea? The program that registers a condition handler could have an area described in its linkage section where it will put information just before it signals a condition. It could put the ADDRESS of that area into this four character field when it registers the handler. Then your handler can use the address to connect to that information. Now there's no limit to the amount of information that can be provided to a handler!



*(Continued on page 15)*

---

# COBOL/370 and LE/370, Article Six

*(Continued from page 14)*

That's how our SAMPHAND example handler will use this field (full sample code is attached at the end of the article):

```
PROGRAM-ID.          SAMPHAND.
                      ...
LINKAGE-SECTION.
  01 CONDITION-TOKEN.
      05 CONDITION-FIRST-EIGHT      PIC X(8).
      05 CONDITION-ISI              PIC X(4).
  01 INFORMATION-ADDRESS            USAGE IS POINTER.
  01 REQUESTED-ACTION              PIC S9(9) BINARY.
  01 NEW-CONDITION                  PIC X(12).

      01 PROGRAM-INFORMATION        PIC X(80).

PROCEDURE DIVISION USING CONDITION-TOKEN, INFORMATION-ADDRESS,
                        REQUESTED-ACTION, NEW-CONDITION.
```

Now when the handler is invoked, it can check the condition token to see if it's one that it should handle. To make the token a little easier to deal with, we might want to "decode" it first. We use another Callable Service for this:

```
CALL 'CEEDCOD' USING CONDITION-TOKEN,
                     SEVERITY1, MESSAGE-NUMBER,
                     CASE, SEVERITY2, CONTROL,
                     FACILITY-ID, ISI-INFO, FC.
```

This service is used to DeCODE the information in a condition token.

CONDITION-TOKEN      (12 byte character)  
Encoded condition token is passed here.

*(Continued on page 16)*

---

# COBOL/370 and LE/370, Article Six

*(Continued from page 15)*

It returns the following:

SEVERITY1	(2 byte binary integer)
The severity associated with the token.	
MESSAGE-NUMBER	(2 byte binary integer)
The number to be combined with the facility ID for messaging	
CASE	(2 byte binary integer)
Always a 1 for user built tokens.	
SEVERITY2	(2 byte binary integer)
Same value as SEVERITY1.	
CONTROL	(2 byte binary integer)
Value of 1 if it is an IBM originated facility ID (such as CEE, IGY)	
Value of 0 if user supplied facility ID (must start with 'H' or greater)	
FACILITY-ID	(3 byte character)
Facility ID to be combined with message number for messaging	
ISI-INFO	(Fullword binary integer)
Value 0 if none.	
FC	(12 byte character)
Condition returned from CEEDCOD service.	

If it is one we want to handle, get the information from the program that signaled the condition:

SET ADDRESS OF PROGRAM-INFORMATION TO INFORMATION-ADDRESS.

Once your handler has “done its thing” (Sent messages? Fixed data in the passed information? Decided to pass on this token?) it needs to set up the return fields for the Condition Manager. Let’s say it decides to “resume” execution in the original program:

MOVE +10 TO REQUESTED-ACTION.

The value for PERCOLATE is +20; PROMOTE is +30. If the decision is to promote, don’t forget to construct a new token and place it in the NEW-CONDITION parameter.

Now, how did the original program register our SAMPHAND condition handler? It uses something called a PROCEDURE POINTER. Previously, we used a Pointer to pass around the address of data in the linkage section.

*(Continued on page 17)*



---

# COBOL/370 and LE/370, Article Six

*(Continued from page 16)*

A procedure pointer is used to pass around the address of a program instead. So, first the original program must define both pointers:

```
WORKING-STORAGE SECTION.  
    01 HANDLER-ADDRESS          USAGE IS PROCEDURE-POINTER.  
    01 INFORMATION-ADDRESS      USAGE IS POINTER.
```

Now load it with an address. We do that with the ENTRY special name:

```
SET HANDLER-ADDRESS TO ENTRY 'SAMPHAND'.
```

If you compile with the DYNAM option, this address will be supplied at run time. If you use NODYNAM, you will need to INCLUDE the handler in your link edit process to make it part of the resulting load module.

Now that we've got the address of the condition handler, let's take care of that information address we want to pass. Of course we need the area in linkage where the program will later put the information:

```
LINKAGE-SECTION.  
    PROGRAM-INFORMATION          PIC X(80).
```

And we need to capture its address:

```
SET INFORMATION-ADDRESS TO ADDRESS OF PROGRAM-INFORMATION.
```

Now we're ready to register the handler:

```
CALL 'CEEHDLR' USING HANDLER-ADDRESS, INFORMATION-ADDRESS, FC.
```

We use the Callable Service CEE(LE/370) HanDLer Register to contact the Condition Manager. We pass it the address of the SAMPHAND program and the address of the data in linkage we want made available to it. And, optionally we include the Feedback Code parameter, depending on whether we want to check the results ourselves, or let the Condition Manager deal with it.

*(Continued on page 18)*

---

# COBOL/370 and LE/370, Article Six

*(Continued from page 17)*

## Signaling a Condition

We've got our condition handler written and registered. Our program is banging along, doing its thing, when suddenly it sees a problem (invalid data or some such). How does it signal the condition to the condition handler?

First, we have to build a condition token. We use another Callable Service for this:

```
CALL 'CEENCOD' USING SEVERITY1, MESSAGE-NUMBER,  
                     CASE, SEVERITY2, CONTROL,  
                     FACILITY-ID, ISI-INFO,  
                     RESULTING-TOKEN, FC.
```

This service is used to eNCODE the information into a condition token. See the CEEDCOD information above for the meanings of the parameters.

We've used NCOD and DCOD here just to show how these tokens can be built. However, a much better approach is to build them once and save them in a COPY member that has 88 level names for all of them. Something like this:

```
88 DIS0123      VALUE X'0002007B50C49E20'.  
88 DIS0124      VALUE X'00020 ...  
... ETC ...
```

Then COPY this member after your definition of the first eight characters of the condition token. When it's time to signal this condition, you just:

```
SET DIS0123 TO TRUE.
```

The handler can then do the same thing. And instead of using DCOD, it can just say:

```
IF DIS0123 ...
```

*(Continued on page 19)*

---

# COBOL/370 and LE/370, Article Six

*(Continued from page 18)*

In any event, we now have our token. We would move any pertinent data to the program-information field in the linkage area, then we're ready to signal the condition:

```
CALL 'CEESGL' USING CONDITION-TOKEN, QDATA, FC.
```

The QDATA parm is an optional full word binary integer that can be used in conjunction with the ISI info. Our example doesn't use it, and just leaves it with a **0** value.

See how simple? I know, I know ... but it really isn't that bad once you start using it. Full example code attached.

When the program that registers a handler is ready to finish, it should unregister the handler. Can you guess?

```
CALL 'CEEHDLU' USING HANDLER-ADDRESS, FC.
```

Yep. "HanDLer Unregister". Technically, you don't HAVE to unregister it, but remember, the Condition Manager passes around the "termination imminent" token before it shuts down the process (even with normal shut down). Do you want your handler being invoked for that? No? Then unregister it before shut down!

Some of you have probably thought of this: What if the **handler** has a bug that triggers a condition? Oops. COBOL/370 doesn't handle recursion well, and that's just what you're going to get. The new condition will trigger the Condition Manager, who will let your condition handler have first crack, which will hit the same bug again, which will trigger another condition, which will ... well, you get the idea.

Fortunately there's a limit to this silliness. The DEPTHCONDLMT (depth of nested conditions limit -- default value of 10) run time option will kick in and cause an ABEND.

The next article will discuss the ability to use information from condition tokens to send messages.

More information on CONDITION HANDLING can be found in:  
IBM SAA AD/Cycle Language Environment/370 Programming Guide (SC26-4818)

*(Continued on page 20)*

# COBOL/370 and LE/370, Article Six

(Continued from page 19)

```
*****
ID DIVISION.
*****

PROGRAM-ID. SAMPHAND.
AUTHOR.      GARY DUFFIELD.
*   THIS PROGRAM IS USED TO TEST COBOL/370 condition
*   handling. It will respond to a condition signaled
*   by ORIGPGM. It will fix the data and resume.

*****

ENVIRONMENT DIVISION.
*****

*****

DATA DIVISION.
*****

WORKING-STORAGE SECTION.

01  FC                                pic x(12).
01  feedback-first-eight redefines FC
                                pic x(8).
    88  CEE000          VALUE X'0000000000000000'.

01  token-fields.
    05  severity1          pic s9(4) binary.
    05  message-number     pic s9(4) binary.
    05  case               pic s9(4) binary.
    05  severity2          pic s9(4) binary.
    05  cntrol             pic s9(4) binary.
    05  facility-id        pic xxx.
    05  isi-info           pic s9(9) binary.

*****

LINKAGE SECTION.
*****

01  condition-token          pic x(12).
01  information-address      usage is pointer.
01  requested-action         pic s9(9) binary.
    88  resume               value +10.
    88  percolate            value +20.
    88  promote              value +30.
01  new-condition            pic x(12).

01  program-information.
    05  name-in              pic x(20).
    05  id-no                pic 9(9).
```

(Continued on page 21)

# COBOL/370 and LE/370, Article Six

(Continued from page 20)

```
*****
PROCEDURE DIVISION using condition-token, information-address,
                        requested-action, new-condition.
*****
main-routine.
*****

*   Decode the condition token
    call 'CEEDCOD' using condition-token,
                        severity1, message-number,
                        case, severity2, cntrol,
                        facility-id, isi-info, FC.

*   If decode fails, promote the FC as the new condition token
    if not cee000
        move FC to new-condition
        set promote to true
    else
*   Check if it is a token we want to handle
        if facility-id = 'DIS'
            evaluate true
                when message-number = 123
                    set address of program-information to
                        information-address
                    pretend we use the name-in field to look up their
*   correct id-no in a table.
*   Suppose the correct id-no is all 8's:
                    move '888888888' to id-no
                    set resume to true
*   if some other token then percolate it
                    when other
                        set percolate to true
                    end-evaluate
                end-if
            end-if.
        goback.
```

(Continued on page 22)

# COBOL/370 and LE/370, Article Six

(Continued from page 21)

```
*****
ID DIVISION.
*****

PROGRAM-ID. ORIGPGM.
AUTHOR. GARY DUFFIELD.
* THIS PROGRAM IS USED TO TEST COBOL/370 condition
* handling. It will register a condition handler and
* then signal a condition.
EJECT
*****

ENVIRONMENT DIVISION.
*****

INPUT-OUTPUT SECTION.

FILE-CONTROL.
    SELECT FILE-IN ASSIGN UT-S-INPUT.
    SELECT FILE-out ASSIGN UT-S-OUTPUT.

*****

DATA DIVISION.
*****

FILE SECTION.

* THIS IS THE INPUT FILE

FD FILE-IN
    RECORDING MODE IS F
    LABEL RECORDS ARE STANDARD
    RECORD CONTAINS 80 CHARACTERS
    BLOCK CONTAINS 0 RECORDS
    DATA RECORD IS INput-record.

01 INput-record.
    05 name-in pic x(20).
    05 id-no-in pic 9(9).
    05 salary-in pic 9(7)v99.

FD file-out
    RECORDING MODE IS F
    LABEL RECORDS ARE STANDARD
    RECORD CONTAINS 80 CHARACTERS
    BLOCK CONTAINS 0 RECORDS
    DATA RECORD IS output-record.
```

(Continued on page 23)

# COBOL/370 and LE/370, Article Six

(Continued from page 22)

```
01  output-record                      pic x(80).

WORKING-STORAGE SECTION.

01  FC                                pic x(12).
01  feedback-first-eight redefines FC
                                pic x(8).
    88  CEE000                      VALUE X'0000000000000000'.

01  storage-fields.
    05  heap-id                      pic s9(9) binary value 0.
    05  storage-needed               pic s9(9) binary value 80.
    05  storage-address              usage is pointer.

01  condition-token-fields.
    05  severity1                    pic s9(4) binary.
    05  message-number               pic s9(4) binary.
    05  case                          pic s9(4) binary value 1.
    05  severity2                    pic s9(4) binary.
    05  cntrol                       pic s9(4) binary value 0.
    05  facility-id                  pic xxx          value 'DIS'.
    05  isi-info                     pic s9(9) binary value 0.
    05  condition-token              pic x(12).

01  condition-handler-field.
    05  handler-address              usage is procedure-pointer.
    05  information-address           usage is pointer.

01  SWITCHES.
    05  EOF-SWITCH                   PIC XXX          VALUE 'NO '.
        88  END-OF-input              VALUE 'YES '.

*****
LINKAGE SECTION.
*****

01  program-information              pic x(80).
```

(Continued on page 24)

# COBOL/370 and LE/370, Article Six

(Continued from page 23)

```
*****
PROCEDURE DIVISION.
*****

main-routine.
*****

*   get some storage for program-information
    call 'CEEGTST' using heap-id, storage-needed,
                        storage-address, FC.
    set address of program-information to storage-address.

*   register the handler
    set handler-address      to entry 'SAMPHAND'.
    set information-address to address of program-information.
    call 'CEEHDLR' using handler-address, information-address,
                        FC.

*   open the files and get the first record
    OPEN INPUT  FILE-IN
        output file-out.
    READ FILE-IN
        AT END set end-of-input to true.

*   main loop
    PERFORM UNTIL end-of-input

*       look for bad id number
        if id-no-in = '9999999999'

*           move input record to linkage, build condition token
            move input-record to program-information
            move 2 to severity1
                severity2
            move 123 to message-number
            call 'CEENCOD' using severity1, message-number,
                                case, severity2, cntrol,
                                facility-id, isi-info,
                                condition-token, FC

            end-call

*           signal the condition
            call 'CEESGL' using condition-token, FC
            end-call
```

(Continued on page 25)



---

# COBOL/370 and LE/370, Article Six

(Continued from page 24)

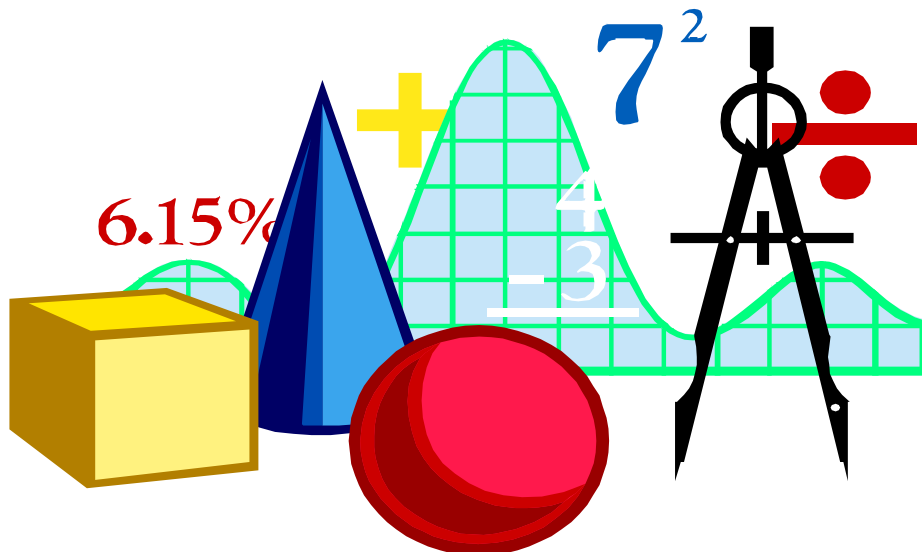
```
*      move the corrected id from linkage back to input buffer
      move program-information to input-record
      end-if

*      write the output, read next input
      move input-record to output-record
      write output-record
      READ FILE-IN
      AT END set end-of-input to true
      end-read
      end-perform

*      close files, unregister handler and shut down
      CLOSE FILE-IN
      file-out.
      call 'CEEHDLU' using handler-address, FC.
      STOP RUN.
```

The remaining articles of this series will be published in future issues of the *DIS Technical Broadcast*.

If you have any questions about these articles, please contact Gary Duffield through the Help Desk at (360) 753-2454. If you would like to obtain a copy of all eight, contact Charlie Martin at (360) 902-3112.



---

# Natural Getmain Threads in CICS

 by Ian Heath

**C**ICS/Database support is currently using what are called “program threads.” Program threads are data assembled together and linked as programs by Natural as a work area. The storage for a program thread is obtained from shared storage and is not protected by the transaction isolation feature offered by CICS 4.1.

Benefits provided by transaction isolation feature are:

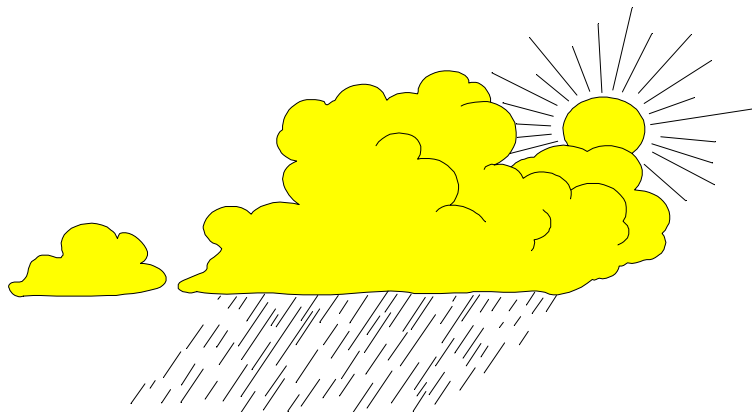
- reducing system outages
- protecting application data
- protecting CICS from applications that pass invalid addresses, and
- aiding application development

Using the Natural Getmain threads we would enjoy the benefits of the transaction isolation feature. To use Getmain threads with this feature, we would identify all transactions associated with the Natural application, assign a thread group of TYPE=GETM, assign each of the transactions a TCLASS value, and turn on the transaction isolation feature for the CICS region(s) in which the application operates.

A reassuring point to be made here is that to go back to the old program threads would NOT require a recycle of the CICS region(s). A rename of one Natural CICS interface module, the control block module, and a CICS new copy of the same module is all that is required.

CICS/Database support is currently in the final stages of a project that will enable the CICS regions to utilize the transaction isolation feature. A *Technical bulletin* will be sent out to discuss the use of the transaction isolation feature.

If you have any questions about program threads, please call Ian Heath, S/390 ADABAS Support at (360) 902-3109.



---

# DFSORT Year 2000 Enhancements

 by Pete Marshall

**D**epartment of Information Services (DIS) is in the process of applying enhancements to DFSORT which will add new features designed to make year 2000 processing easier. These enhancements will allow users to set a 'century window' in DFSORT, as well as allowing for two-digit year processing.

The following information is condensed from the IBM documentation that accompanied the enhancement.

## SETTING THE CENTURY WINDOW FOR TWO-DIGIT YEARS

A new option will allow you to specify the century window you want DFSORT to use for interpreting two-digit years. A century window spans a 100 year period, and is used to control how the two-digit years 00 to 99 are interpreted. For example, 56 will be interpreted as 1956 if you set the century window to 1950-2049 or as 2056 if you set the century window to 1990-2089.

DFSORT will allow you to set either a fixed century window or a sliding century window as follows:

- **Y2PAST=f** specifies a fixed century window starting at **f**, where **f** can be any value between 1000 and 3000. For example, Y2PAST=1985 will always start the century window at 1985, resulting in a century window of 1985-2084.

You might want to use a fixed century window for two-digit year data that you started collecting in some particular year (for example, 1985 in this case).

- **Y2PAST=s** specifies a sliding century window starting **s** years before the current year, where **s** can be any value between 0 and 100. For example, if the current year is 1996, Y2PAST=80 starts the century window at 1996 - 80 or at 1916, resulting in a century window of 1916-2015. When 1997 rolls around, Y2PAST=80 starts the century window at 1917 (1997-80), which sets the century window to 1917-2016. **Y2PAST=s** causes the century window to auto-matically slide year by year.

You might want to use a sliding century window for two-digit year data you collect over a rolling time period.

Thus, the Y2PAST option allows you to control how DFSORT interprets the two-digit years 00 to 99 for various sets of data.

*(Continued on page 28)*

---

# DFSORT Year 2000 Enhancements

*(Continued from page 27)*

**NOTE:** The DIS installation default will be Y2PAST=80 which sets a sliding window starting 80 years before the current year. You can use the Y2PAST run-time option to override this for different applications.

## SELECTING NEW FORMATS FOR TWO-DIGIT YEAR DATES

DFSORT will provide several new formats that will help you deal with a wide variety of dates containing two-digit years.

You will be able to use the new Y2C, Y2Z, Y2P and Y2D formats to identify your two-digit character, zoned decimal, packed decimal and decimal year data, respectively. DFSORT will correctly sort, merge and transform two-digit years identified with these formats as if they were four-digit years, using your specified fixed or sliding century window.

You will be able to use the new PD0 format to identify your two-digit packed month data and two-digit and three-digit packed day data. DFSORT will correctly sort, merge and transform months and days identified with these formats.

## SORTING AND MERGING ON TWO-DIGIT YEAR DATES

You will also be able to use DFSORT's new formats (Y2C, Y2Z, Y2P, Y2D and PD0) to correctly sort and merge many different kinds of character, zoned decimal and packed decimal dates, including those with two-digit years, according to the century window you specify.

For each sort or merge application involving two-digit years, you need to decide whether to use DIS's default century window (sliding window, current year minus 80) or set a different century window for the application using the Y2PAST run-time option.

## TRANSFORMING TWO-DIGIT YEAR DATES TO FOUR-DIGIT YEAR DATES

You will be able to use DFSORT's new formats (Y2C, Y2Z, Y2P, Y2D and PD0) to transform many different kinds of dates, including those with two-digit years, according to the century window you specify. Using the OUTREC operand of the OUTFIL statement, you will be able to:

- convert two-digit year fields in character, zoned decimal and packed decimal format to four-digit year fields in character format (for example, convert C'yy' to C'yyyy')

*(Continued on page 29)*

---

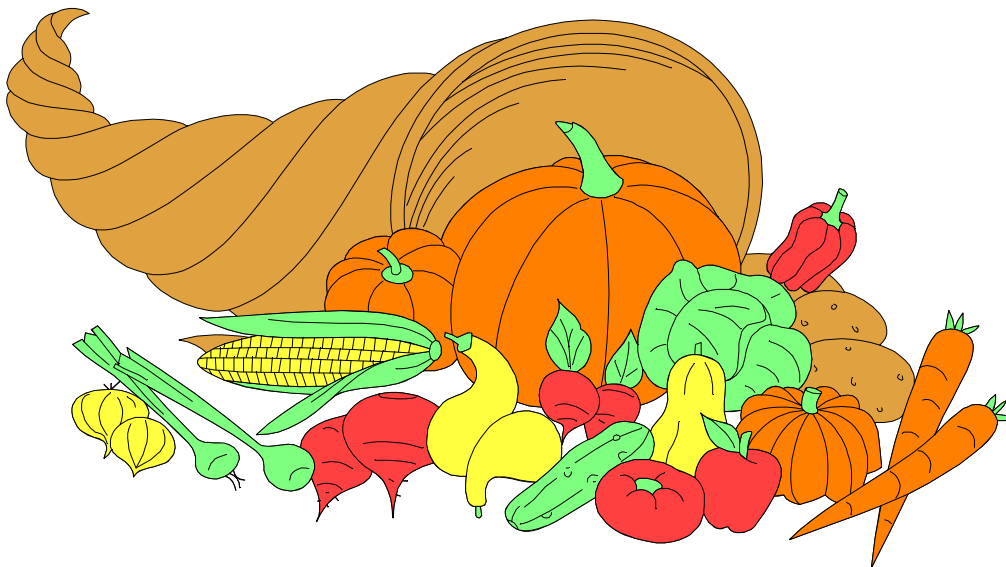
# DFSORT Year 2000 Enhancements

*(Continued from page 28)*

- convert month and day fields in zoned decimal and packed decimal format to month and day fields in character format (for example, convert P'ddmmyy' to C'ddmmyyyy')
- rearrange the order of the year, day and month (for example, convert Z'mmddyy' to C'yyyymmdd')
- insert separators between the year, month and day (for example, convert P'yyddmm' to C'dd.mm.yy' or C'mm/dd/yyyy')

For more information on DFSORT, visit IBM's DFSORT web site at:

**<http://www.storage.ibm.com/software/sort/srtmhome.htm>**, or call (360) 753-2454 and ask to be connected to Pete Marshall.



---

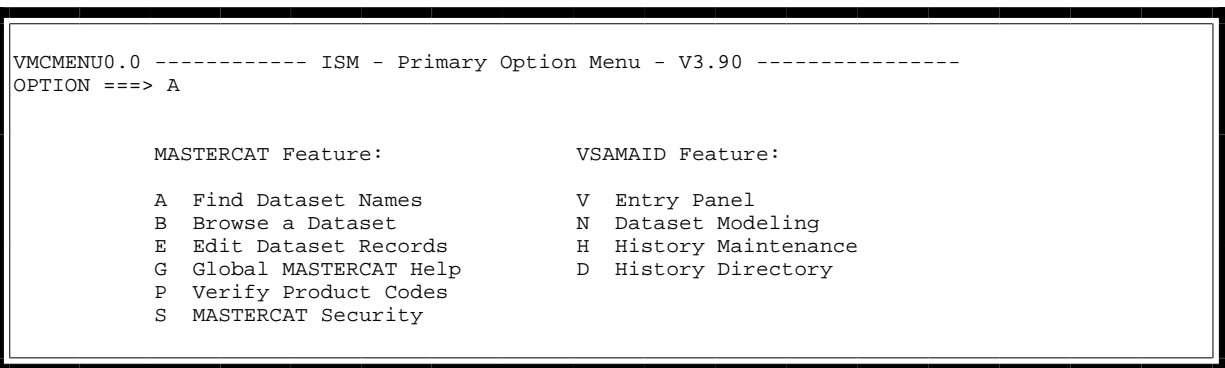
# How to Find Unused GDG Base Catalog Entries Using MASTERCAT

 by Jim Collinsworth

**M**ASTERCAT is a powerful tool for querying the ICF catalogs. It provides information and functions not readily available through IDCAMS or other utilities. Using MASTERCAT is a fast and easy way to find and delete unused Generation Data Group (GDG) bases, thereby speeding up your catalog searching and job processing. To find and delete old GDG bases with no active generation data sets:

- from the Main Menu, select item **S** for Storage, press the **ENTER** key and the Storage Utilities Menu will be displayed.
- select MC for MASTERCAT and press the **ENTER** key.
- from the ISM Primary Option Menu (Figure 1), select option **A** for Find Dataset Names and press the **ENTER** key.

Note: Options B, E, P and S are not available for your use.



```
VMCMENU0.0 ----- ISM - Primary Option Menu - V3.90 -----
OPTION ==> A

MASTERCAT Feature:          VSAMAID Feature:
A Find Dataset Names        V Entry Panel
B Browse a Dataset          N Dataset Modeling
E Edit Dataset Records      H History Maintenance
G Global MASTERCAT Help     D History Directory
P Verify Product Codes
S MASTERCAT Security
```

Figure 1

Next the MASTERCAT Data Set Locator screen (Figure 2) is displayed on page 33. This is the screen where you fill in your search options. There are four fields on this screen:

(Continued on page 31)

---

# How to Find Unused GDG Base Catalog Entries Using MASTERCAT

(Continued from page 30)

**Entry Name** - used to specify the names of entries for MASTERCAT to find in catalogs. The name - selection conventions are:

**Full data set name**

*Example:*

LEVEL1.LEVEL2.FILE

**Partial data set name**

Data set name with any combination of the four masking characters

+ - individual characters

\* - levels or nodes

‘\*\*\*’ or ‘++’ indicates match anything beyond this point

a character string enclosed in quotes indicates match anywhere in a name

*Example:*

LEVEL+.FILE	selects LEVEL1.FILE or LEVEL2.FILE
LEVEL1.*	selects LEVEL1.FILE or LEVEL1.DATA
LEVEL1.**	selects everything with the 1st node of LEVEL1
LEVEL1.++	selects everything with the 1st node of LEVEL1
LEVEL1.*.'JCL'	selects all data sets with LEVEL1 in the 1st node, anything in the 2nd node and JCL anywhere after the second node

**IMPORTANT NOTE:** Never put the characters ‘\*\*\*’ or ‘++’ or the word ‘ALL’ by itself in the ‘Entry Name’ field. This is a generic data set name search of all catalogs and all data sets. This activity involves a significant amount of overhead and can be costly.

**Catalog Name** - used to specify the names of the catalogs MASTERCAT will need to search. This entry will be left blank.

**Search Items** - further refines the selection process to look for entries in catalog data fields which have certain values. The variables are defined as follows:

fieldname - the name of a catalog data field.

(Continued on page 32)

---

# How to Find Unused GDG Base Catalog Entries Using MASTERCAT

*(Continued from page 31)*

operator - the operator for the type of comparison to make between the fieldname and the value. The operator may be one of the following:

- = fieldname equal to value
- ≠ fieldname not equal to value
- > fieldname greater than value
- < fieldname less than value

value - a value against which to test the contents of the fieldname.

Use AND or OR to indicate the logical connection between two fieldname operator value expressions.

**Show Items** - allows you to display specific catalog information.

For more information, move your cursor to the line of the item you would like help with and press your PF1 key. Help screens will appear which will give you the information you are requesting. For a list of search or show items, press the PF4 key.

## How to find unused GDG base definitions.

Fill in the MASTERCAT Data Set Locator screen with:

- Entryname(s) => your generation data set(s) names or patterns
- Search item(s) => **(TYPE = GDG) AND (GATCNT = 0)**  
This criteria will search for catalog entry type equal to generation data set and active generation count equal to zero.
- Show item(s) => **CREATION**  
Creation will show either the date the GDG base was created or the date the last generation was created.

*(Continued on page 33)*



# How to Find Unused GDG Base Catalog Entries Using MASTERCAT

(Continued from page 32)

The following is a sample screen:

```
MFINDER .3 ----- MASTERCAT - Data Set Locator - V3.90 -----
COMMAND ===>

Entry Name    ==> DATA123.DSK.BKUP.**
Catalog Name  ==>
Search Items  ==> (TYPE = GDG) AND (GATCNT = 0)

Show Items   ==> CREATION
```

Figure 2

The results of the search are displayed in the MASTERCAT Data Set Selection (Figure 3) screen. The entries are listed within each user catalog. To delete the unused GDG base, type the word **DELETE** on the line to the left of the GDG base entry and press the **ENTER** key.

```
MPICKER .7 ----- MASTERCAT - Data Set Selection - V3.90 ----- ROW 0001 OF 0038
COMMAND ===> SCROLL ==> PAGE
Items: 1-3 of 3

Current Catalog: ICFCAT.USER.AGY155
COMMAND ENTRY-NAME Creation-Date Type Gatcnt
-----
CATALOG: ICFCAT.USER.AGY155
DATA123.DSK.BKUP.ESADLB FRI 02 APR 1993 092 GDG 0
DATA123.DSK.BKUP.ESARES FRI 02 APR 1993 092 GDG 0
DATA123.DSK.BKUP.M28RES FRI 02 APR 1993 092 GDG 0
DELETE DATA123.DSK.BKUP.PANVL2 FRI 02 APR 1993 092 GDG 0
DATA123.DSK.BKUP.PAN901 FRI 02 APR 1993 092 GDG 0
DATA123.DSK.BKUP.PSYS01 FRI 02 APR 1993 092 GDG 0
DATA123.DSK.BKUP.PSYS02 FRI 02 APR 1993 092 GDG 0
DATA123.DSK.BKUP.PSYS03 FRI 02 APR 1993 092 GDG 0
```

Figure 3

(Continued on page 34)

# How to Find Unused GDG Base Catalog Entries Using MASTERCAT

(Continued from page 33)

The MASTERCAT Interactive IDCAMS screen (Figure 4) shows the IDCAMS command which will be executed. To execute the command press PF5, and to cancel press PF3.

```
MAMS      .a ----- MASTERCAT - Interactive IDCAMS - V3.90 ---- ROW 0001 OF 0005
COMMAND ===>                                SCROLL ===> CSR
                                                COL 0001 OF 0132

CMD  IDCAMS PARAMETERS INPUT AREA
-----
-    DELETE -
-    (DATA123.DSK.BKUP.PANVL2) -
-    GDG      -
-    NOFORCE  -
-    NOPURGE
```

Figure 4

A confirmation screen (Figure 5) is displayed after the deletion. If you receive an error message, press PF1 for more information.

```
MAMS      .5 ----- MASTERCAT - Interactive IDCAMS - V3.90 ---- ROW 0001 OF 0011
COMMAND ===>                                SCROLL ===> CSR
MAXIMUM CONDITION CODE WAS 0                  COL 0001 OF 0132
CMD  IDCAMS PARAMETERS INPUT AREA
-----
IDCAMS  SYSTEM SERVICES                                TIME:

-    DELETE -
-    (DATA123.DSK.BKUP.PANVL2) -
-    GDG      -
-    NOFORCE  -
-    NOPURGE
IDC0550I ENTRY (B) DATA123.DSK.BKUP.PANVL2 DELETED
IDC0001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS  0

IDC0002I IDCAMS PROCESSING COMPLETE. MAXIMUM CONDITION CODE WAS  0
```

Figure 5

MASTERCAT is a powerful catalog query tool with multiple uses. The Storage Management group will document other uses of this tool in upcoming *Technical Broadcast*. If you have any questions, please call Jim Collinsworth at (360) 902-3588.

---

# Billing of Tape and Disk Data Sets in the S/390 MVS Environment

 by Kimberly Starkey

## What Reports are Provided by Storage Management?

**D**epartment of Information Services (DIS) Storage Management produces two reports for all agencies that utilize Tape and Disk storage. These reports provide various information including which work authorization numbers or billing account codes are used to accumulate charges.

The reports are distributed monthly, on-line, through the Report Distribution System (RDS) and in printed form. The Tape Utilization Report has three sections: local round for 9-track tape, local square for 18/36-track tape and remote offsite square for tapes registered to the offsite service. The Tape Utilization Report contains the volser, expire date, creating job name, data set name, create date, out of area, programmer name, days in and accumulated cost in dollars. The Disk Utilization Report contains the volser, delete status, data set name, storage type, create date, used date, allocated tracks, used tracks, accumulated mega-byte hours and accumulated cost in dollars.

## To Calculate the Cost of Tape and Disk Storage.

The accumulated charges for tape media are based on tape days. The rates are .096 for 9-track (round reel), .08 for 18/36 track (cartridge) and .14 for remote offsite tapes. When the tape media is expired the agency no longer accumulates charges. The only exception is if the tape is checked out to the agency and given an OUTCODE of SHRT. Even though the tape may have reached its expiration date the agency will continue to accumulate charges until it is returned and checked back in.

Example: To calculate the charges of an 18 track cartridge for 7 days:

$$7 * .08 = .56 \text{ total of 56 cents.}$$

The accumulated charges for disk storage are based on mega byte hours. The rate is .0008. You may note on the report that a disk data set is listed more than once. This occurs if the data set is deleted and re-created by your process or a storage management process. The billing reflects the amount of time the data set resided on each of the volumes.

*(Continued on page 36)*

---

# Billing of Tape and Disk Data Sets in the S/390 MVS Environment

*(Continued from page 35)*

Example: To calculate the charges for 1 cylinder for 30 day month

1 cylinder is 15 tracks

1 track is 58786 bytes

15 tracks is 881790 bytes

881790 bytes divided by one million is .88 megabytes (MB)

.88 megabytes \* 30 days \* 24 hours \* .0008 = .51 cents

## **To Control Where Tape and Disk Data Sets Accumulate charges.**

Tape and disk storage accumulate charges based on entries in a table maintained by the Storage Management group. The billing account code on your job card is not used. To review how your storage is charged, Storage Management provides three files per agency. These files can be accessed on-line through the TSO Data Set List Utility option (3.4). These files are updated every Monday morning and will reflect changes made for the past week.

The files are:

FBIL155.PPATTERN.AGYxxx      where XXX is your agency number.

This file contains all the entries used by the metering process. To reduce the number of entries in the table there are three masking characters available.

? - one character

\* - remainder of the node

/ - anything after this

FBIL155.PTAPE.DEFAULT.AGYxxx

FBIL155.PDISK.DEFAULT.AGYxxx

These files contain all the data sets that were not charged to a specific billing account code. When a data set does not match an existing pattern, the agency default account code is used. To remove these data sets from the default billing account, an account code is needed to be assigned to the data set name, so an entry can be added to the billing table.

*(Continued on page 37)*

---

# Billing of Tape and Disk Data Sets in the S/390 MVS Environment

(Continued from page 36)

## To Provide Information to Storage Management.

When a new high level qualifier (HLQ) is defined, notify Storage Management of the billing code. Also, when a HLQ is taken out of service, it should be deleted from the billing account table.

To make additions, deletions, and instructions on how to condense the entries, contact or send updates to S/390 Storage Management section via MSMail: DIS STORMGMT (within DIS), FAX: (360) 586-8992, Campus Mail: 42445 or to an exchange file on-line.

To provide an exchange file, copy entries from the FBIL155.PPATTERN.AGYxxx file to your exchange file and make the appropriate changes. Notify us when the file is ready and we will replace the entries in original billing account table. You must follow the format of the FBIL155.PPATTERN.AGYxxx file.

Column	Description
1	Single Quote
2 - 6	Agency and SubAgency Number
7	Blank Space Required
8 - 13	Account Number
14 - 58	Data Set Name
59	Single Quote
60	Blank Space Required
61 - 67	Last Name or Group Adding Entry
68 - 72	Julian Date when entry was added

Example:

'15500 000000DATA.SET.NAME./                    ' STMGMT 96278

If you need assistance or have any questions, please contact Storage Management at (360) 902-3588.

# the Technical Broadcast



Charie L. Martin, Editor  
Department of Information Services  
Adams Building  
1310 Jefferson ST SE  
Mail Stop: 42445  
Olympia, WA 98504-2445

***Technical Broadcast Staff*** - The *Technical Broadcast* is published quarterly by the Department of Information Services (DIS). The purpose is to provide a forum for customer information-sharing of upcoming system enhancements and optimization tips. We invite your articles, comments, and suggestions.

DIS is an equal opportunity employer and does not discriminate on the basis of race, religion, color, sex, age, nationality, or disability.